

# Robot Swarms as Hybrid Systems: Modelling and Verification

Stefan Schupp

TU Wien, Vienna, Austria

Francesco Leofante

Imperial College London, London, United Kingdom

Leander Behr

RWTH Aachen University, Aachen, Germany

Erika Ábrahám

RWTH Aachen University, Aachen, Germany

Armando Taccella

University of Genoa, Genoa, Italy

A swarm robotic system consists of a team of robots performing cooperative tasks without any centralized coordination. In principle, swarms enable flexible and scalable solutions; however, designing individual control algorithms that can guarantee a required global behavior is difficult. Formal methods have been suggested by several researchers as a mean to increase confidence in the behavior of the swarm. In this work, we propose to model swarms as hybrid systems and use reachability analysis to verify their properties. We discuss challenges and report on the experience gained from applying hybrid formalisms to the verification of a swarm robotic system.

## 1 Introduction

Swarm robotic systems are distributed systems wherein a set of robots cooperatively perform a task, without any centralized coordination [29]. Although individual robots are governed by relatively simple reactive controllers, interactions within the swarm may give rise to complex behaviors that were not explicitly programmed. Ultimately, these behaviors enable the swarm to achieve goals that would defy each single robot in isolation, or would require more expensive robots to achieve the same goals as effectively as the swarm does – see, e.g., [4, 33] for some examples.

While understanding individual robot behavior is easy, predicting the overall swarm behavior is difficult, and thus engineering controllers for individual robots that will guarantee a desired swarm behavior is not a straightforward task. Traditionally, the analysis of swarms is carried out either by testing real robot implementations, or by computational simulations [20, 22]; however, these approaches provide little guarantees as they suffer from intrinsically incomplete coverage. As suggested by many authors, higher levels of assurance in swarm behavior can be obtained via formal methods [32, 30, 16, 5, 18, 23]. However most approaches abstract away details about the continuous dynamics of the robots, which may indeed be crucial for the emergence of desired behaviors.

In this paper, we fill this gap by investigating the problem of providing assurance guarantees for swarm robotic systems through mixed discrete-continuous formalisms. In particular, we propose to use hybrid automata [14] to define richer models of swarms and employ tools for reachability analysis to overcome the incomplete coverage of testing and simulation. Here we report on the application of reachability analysis of hybrid automata to verify a controller developed for swarm applications. Modeling challenges are discussed, together with features and limitations of tools for reachability analysis. Our contributions can be summarized as follows:

- Applying hybrid automata modeling and reachability analysis as a method to formally engineer robot swarms; to the extent of our knowledge, this is the first contribution in this direction.
- Modeling and analyzing a simple but realistic swarm robotic system entailing synchronization without central coordination.
- Discussing challenges together with new solutions proposed to overcome modeling and scalability issues.

The remainder of this paper is organized as follows. Section 2 introduces background notions on swarm systems and their verification. Section 3 contextualizes the analysis of robot swarms as hybrid systems and Section 4 presents our case study, together with experimental results. Finally, Section 5 provides concluding remarks and future directions of research.

## 2 Background

### 2.1 Swarm Robotic Systems

A robot swarm can be defined as a specific kind of distributed autonomous mobile robotic system wherein a set of robots is meant to perform some kind of collective task [29]. This is achieved following decentralized, behavior-based control rules relying on “social” interactions among the robots.

A swarm can be characterized as in [4]: robots are autonomous; they are situated in an environment and act to modify it; robot’s sensing and communication capabilities are local; robots do not have access to centralized control and/or global knowledge and cooperate to accomplish a task. The cooperation among robots happens in different ways, including explicit communication through, e.g., a wireless network, or implicitly through *stigmergy* [2], i.e., robots sense changes made by other robots, and then adjust their behavior accordingly. In any case, the collective behavior is not predefined to any extent at the global level, but it is most likely to be *emergent*, i.e., the result of several local robot-to-robot and robot-to-environment interactions. While the definition of “emergence” has been the subject of different contributions, here we take as working definition the one given in [25] asserting that emergent behaviors are characterized by two properties: (i) they are manifested by global states or time-extended patterns which are not explicitly programmed in, but result from local interactions among a system’s components; (ii) they are considered interesting based on some observer-established metric.

In this work we study emergent behaviors within swarms of MarXbots [3], mobile robots that have been conceived and built through several European projects focusing on swarm – see, e.g, [27]; successful applications of such robots can be found, e.g., in [10] and [9]. To support our experimentation, we use ARGOS [31], a simulator designed to efficiently simulate complex experiments involving large swarms of robots, developed within the same European projects mentioned above.

### 2.2 Formal Verification of Swarms

To the best of our knowledge, the first contribution along this line of investigation is [32], wherein the authors investigated the applicability of formal methods to the verification and validation of spacecraft using swarm technology. More specifically, they considered a number of approaches including Communicating Sequential Processes (CSP), process algebras, X-machines and Unity Logic. However, their conclusion at the time of the contribution (2004) was that none of the approaches had all the properties required to assure correct behavior and interactions of swarms in the context of the ANTS (Autonomous Nano Technology Swarm) concept mission. In [30], agent-oriented software engineering is investigated to

provide support for the development of swarm robotics systems, still in the context of the ANTS mission. However, no mention related to assuring behaviors is to be found in the contribution which is largely confined to model-based design and implementation techniques. A series of papers by Dixon et al. – see, e.g., [16] for the most recent contribution in the series – explores the potential of modeling robot swarms using a composition of (probabilistic) finite state machines and of proving swarm-level requirements using model checking of (probabilistic) temporal logic. Noticeably, in the case of probabilistic models and logic, the model of each single robot controller is very close to the actual implementation, and model checking of relevant properties is reported to be feasible for swarms of relatively small size – less than 4 robots according to the experiments in [16].

The problem of verifying swarm systems and their emergent properties was also considered more recently in [18, 17, 23, 24], where a number of techniques have been proposed to improve the scalability of existing verification algorithms. However, these approaches abstract away the physical dynamics of the robots of the environment, which may be determinant for the emergence of unforeseen behaviors. In robotics, this is more than just an abstract principle, because the implements are physical agents that can damage the environment, and thus should be subject to stringent requirements – see, e.g., ISO/TC 299 published standards about safety requirements for various kinds of robots. Collective behaviors may sometimes emerge not only from the current state of the single robots, but also from the continuous dynamics in time and space. In order to model such complex properties, discrete transitions as well as continuous dynamics must be included in the formal model. For this reason, in this work we elaborate on the choice of *hybrid automata* as a modeling formalism.

### 3 Hybrid Systems Reachability Analysis

The approaches discussed in the previous section abstract the physical components of robots and simplify their interaction with the environment. Indeed, all robotic systems are a combination of programmed digital controllers and implements – sensors and actuators – interfacing with the physical world. Therefore, an accurate model of robot operation should include both the (discrete) control states and the (continuously) varying physical quantities. In the case of robot swarms, we believe that failing to take into account the physical components, may hamper our ability to determine whether the swarm will behave correctly at all times. *Hybrid automata* are a well-established formalism to model systems combining discrete states and continuously varying quantities, and methods for their verification have been successfully developed.

#### 3.1 Hybrid Automata

**Definition 1 (Hybrid automaton: Syntax [14])** A hybrid automaton  $\mathcal{H} = (Loc, Var, Flow, Inv, Edge, Init)$  is a tuple consisting of:

- A finite set  $Loc$  of locations or control modes.
- A finite ordered set  $Var = \{x_1, \dots, x_n\}$  of real-valued variables; we also use the vector notation  $\vec{x} = (x_1, \dots, x_n)$ . The number  $n$  is called the dimension of  $\mathcal{H}$ . By  $\dot{Var}$  we denote the set  $\{\dot{x}_1, \dots, \dot{x}_n\}$  of dotted variables (which represent first derivatives during continuous change), and by  $Var'$  the set  $\{x'_1, \dots, x'_n\}$  of primed variables (which represent values directly after a discrete change). Furthermore,  $Pred_X$  is the set of all predicates with free variables from  $X$ .
- $Flow : Loc \rightarrow Pred_{Var \cup \dot{Var}}$  specifies for each location its flow or dynamics.
- $Inv : Loc \rightarrow Pred_{Var}$  assigns to each location an invariant.

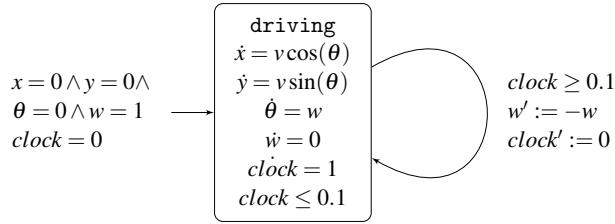


Figure 1: Hybrid automaton for a differential drive robot with constant speed  $v = 1$ .

- $\text{Edge} \subseteq \text{Loc} \times \text{Pred}_{\text{Var}} \times \text{Pred}_{\text{Var} \cup \text{Var}'} \times \text{Loc}$  is a finite set of discrete transitions or jumps. For a jump  $(l_1, g, r, l_2) \in \text{Edge}$ ,  $l_1$  is its source location,  $l_2$  is its target location,  $g$  specifies the jump's guard, and  $r$  its reset function, where primed variables represent the state after the step.
- $\text{Init} : \text{Loc} \rightarrow \text{Pred}_{\text{Var}}$  assigns to each location an initial predicate.

An example hybrid automaton that models a differential drive robot driving a chicane is depicted in Figure 1. The system has 5 variables used for describing the relevant physical quantities of the robot (position, steering and angular velocity). An initial variable assignment (*initial set*) specifies the initial conditions of the system. The automaton has one *control mode* (*driving*) modeling the continuous behavior of the robot which is specified by ordinary differential equations (ODEs). The continuous behavior inside the location is limited by the invariant on the variable *clock*. This variable models a timer, which guards the only transition in the model. As soon as the guard condition  $\text{clock} \geq 0.1$  is satisfied, the transition is *enabled* and can be taken.

The *state* of a hybrid automaton is specified by a pair  $(l, \vec{v}) \in \text{Loc} \times \mathbb{R}^n$  of a location and a variable valuation. The evolution of a hybrid system over time can be described by a run in the respective hybrid automaton, which includes both continuous evolution (*flow*) and discrete state changes (*jump*). The formal semantics of a hybrid automaton is defined as follows.

**Definition 2 (Hybrid automaton: Semantics)** *The operational semantics of a hybrid automaton  $\mathcal{H} = (\text{Loc}, \text{Var}, \text{Flow}, \text{Inv}, \text{Edge}, \text{Init})$  of dimension  $n$  is defined by the following rules:*

$$\frac{l \in \text{Loc} \quad \vec{v}, \vec{v}' \in \mathbb{R}^n \quad f : [0, \delta] \rightarrow \mathbb{R}^n \quad df/dt = \dot{f} : (0, \delta) \rightarrow \mathbb{R}^n \quad f(0) = \vec{v} \quad f(\delta) = \vec{v}' \quad \forall \epsilon \in (0, \delta). f(\epsilon), \dot{f}(\epsilon) \models \text{Flow}(l) \quad \forall \epsilon \in [0, \delta]. f(\epsilon) \models \text{Inv}(l)}{(l, \vec{v}) \xrightarrow{\delta} (l, \vec{v}')} \quad \text{Rule } e_{\text{flow}}$$

$$\frac{e = (l, g, r, l') \in \text{Edge} \quad \vec{v}, \vec{v}' \in \mathbb{R}^n \quad \vec{v} \models g \quad \vec{v}, \vec{v}' \models r \quad \vec{v}' \models \text{Inv}(l')}{(l, \vec{v}) \xrightarrow{e} (l', \vec{v}')} \quad \text{Rule } e_{\text{jump}}$$

A path of  $\mathcal{H}$  is a (finite or infinite) sequence  $(l_0, \vec{v}_0) \xrightarrow{\delta_0} (l_1, \vec{v}_1) \xrightarrow{e_1} (l_2, \vec{v}_2) \xrightarrow{\delta_2} (l_3, \vec{v}_3) \xrightarrow{e_3} (l_4, \vec{v}_4) \xrightarrow{\delta_4} \dots$  with  $(l_i, \vec{v}_i)$  states of  $H$ ,  $\delta_i \in \mathbb{R}_{\geq 0}$ ,  $e_i \in \text{Edge}$ , and  $\vec{v}_0 \models \text{Init}(l_0) \wedge \text{Inv}(l_0)$ . A state  $(l, \vec{v})$  is reachable in  $\mathcal{H}$  if there is a path  $(l_0, \vec{v}_0) \xrightarrow{\delta_0} (l_1, \vec{v}_1) \xrightarrow{e_1} (l_2, \vec{v}_2) \xrightarrow{\delta_2} \dots$  of  $\mathcal{H}$  with  $(l, \vec{v}) = (l_i, \vec{v}_i)$  for some  $i \geq 0$ .

Paths defined over states naturally generalize to paths over state sets  $(l, V) = \{(l, \vec{v}) \mid \vec{v} \in V\}$  for  $V \subseteq \mathbb{R}^n$ : a symbolic path  $(l_0, V_0) \xrightarrow{\delta_0} (l_1, V_1) \xrightarrow{e_1} (l_2, V_2) \dots$  represents the set of all paths  $(l_0, \vec{v}_0) \xrightarrow{\delta_0} (l_1, \vec{v}_1) \xrightarrow{e_1} (l_2, \vec{v}_2) \dots$  with  $\vec{v}_i \in V_i$  for all  $i$ .

The above formalism can be extended with *urgent* jumps, which force the control to take a jump as soon as an urgent jump is enabled. In graphical illustrations, we mark urgent jumps with a star. In our models, in the source locations of urgent jumps no time can pass, thus urgency could be modeled by introducing a fresh clock with derivative 1, resetting it to 0 when entering the location, and adding an invariant stating that the variable value is 0. However, some approaches can analyse reachability more efficiently when using urgent jumps.

To model compositional hybrid systems involving communication between the components, we use the *parallel composition* of hybrid automata. Besides shared variables, we can model communication by annotating jumps with *synchronization labels*. Semantically, time evolves in all components in parallel; a jump of a component can be taken only if all components that have the given label take a jump with that label simultaneously. Thus when building the parallel composition syntactically as a single hybrid automaton, a composed jump needs to be added for each possible combination of synchronizing jumps. Therefore, in general, the size of the composition increases exponentially not only in the number of locations, but also in the number of jumps.

### 3.2 Reachability Analysis

Once a hybrid automaton model of a given hybrid system has been formalized, we are interested in analyzing its behavior. Given that the hybrid systems we are considering are physical agents that can act and modify the environment, we want to enforce stringent safety requirements upon their behavior. Such requirements can be formalized as sets of states to be avoided in the state space of a hybrid automaton.

The *reachability problem* for hybrid automata is the problem to decide whether a given state (or any state from a given set) is reachable in a hybrid automaton. As the reachability problem for hybrid automata is in general undecidable, some approaches aim at computing an *over-approximation* of the set of reachable states of a given hybrid automaton. We focus on approaches based on *flowpipe-construction*, which iteratively over-approximate the set of reachable states by the union of a set of state sets – see e.g. [11] for further details. To *represent* a state set, typically either a *geometric* or a *symbolic* representation is used. Geometric representations specify state sets by geometric objects like boxes [28], (convex) polytopes [40], template polyhedra [34], zonotopes [13], or ellipsoids [19], whereas symbolic representations use, e.g., support functions [21] or Taylor models [7]. These representations might have major differences in the precision of the representation (the size of over-approximation), the memory requirements and the computational effort needed to apply operations like intersection, union, linear transformation, Minkowski sum or test for emptiness.

For a given initial state set  $p$ , flowpipe-construction-based approaches first over-approximate the set of states reachable from  $p$  via time evolution. Time evolution is usually restricted to a *time horizon* (either per location or for the whole execution), which is divided into smaller time segments. The time successors (called the *flowpipe*) from  $p$  are over-approximated by a sequence of state sets  $p_1, \dots, p_k$  (*flowpipe segments*), one for each time segment. For each of these, all possible jump successors are computed and for each jump successor set the whole procedure is repeated iteratively. The algorithm terminates if either a fixed point is detected, or the time horizon is reached, or a maximal number of jumps (*jump depth*) have been considered, or if all successor sets are empty (i.e. there are no jump successors from the flowpipes). The union of all computed state sets over-approximates reachability within the given time and jump bounds. A non-exhaustive list of software implementations of flowpipe-construction-based methods includes CORA [1], FLOW\* [8], HYCREATE [15], HYPY [36] and SPACEEX [12].

The resulting over-approximations for time- and jump-successor states for all reachable paths (over state sets) can be collected in the *reachability tree* in which nodes represent time-evolution and the

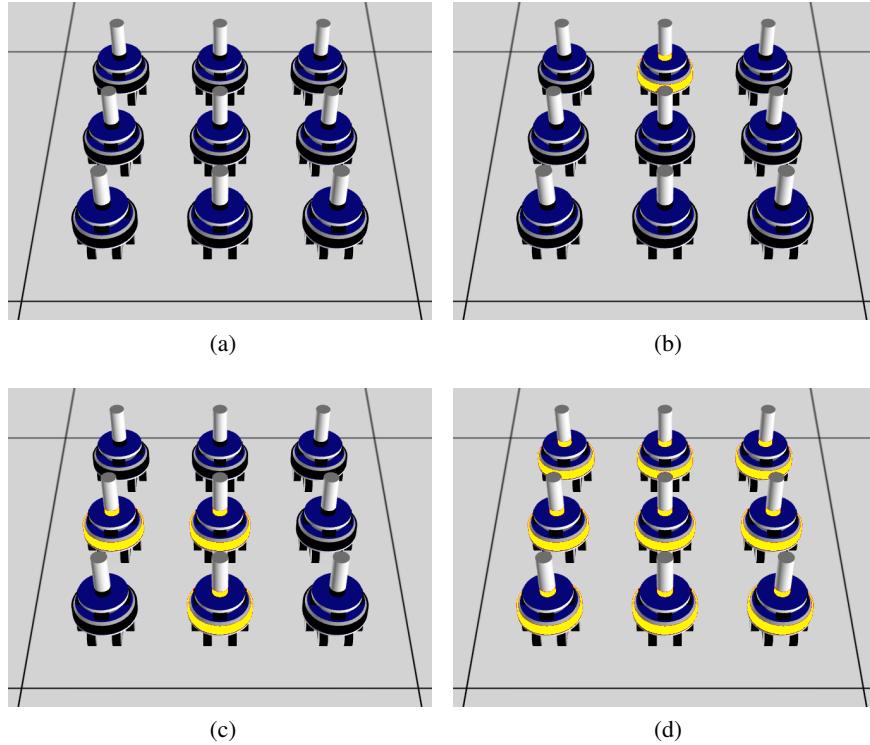


Figure 2: Synchronization in the simulator.

parent-child relation between nodes represents a discrete jump. Note that since the method computes over-approximations of sets of reachable states, consequently only over-approximations of paths over state sets are computed and stored in the reachability tree. Thus it may happen, that the reachability tree contains sub-trees which are not reachable in the hybrid system model, but are only included due to the over-approximation. For further details on the reachability tree we refer to [35].

## 4 Case Study

This section presents our analysis pertaining the application of hybrid systems verification techniques to a case study in swarm robotics. In the following we introduce the problem considered, together with a discussion on modeling and verification in the hybrid setting.

### 4.1 Synchronization without Central Coordination

**Problem description.** Mutual synchronization is a natural phenomenon whereby a population of individuals synchronize over a common behavior and act in perfect unison among themselves – see, [6, 38, 39] for some examples. Interestingly, achieving such mutual synchronization requires a cooperative effort, which is undertaken without any central coordination between individuals. In this experiment we reproduce the behavior of pulse-coupled oscillators as described in [26] and implemented in the ARGoS simulation environment (see Figure 2). The model involves a population of  $n$  MarXbots [3], each of which is equipped with an LED. For  $i \in \{1, \dots, n\}$ , the  $i$ th robot is characterized by a clock  $x_i$  subject to

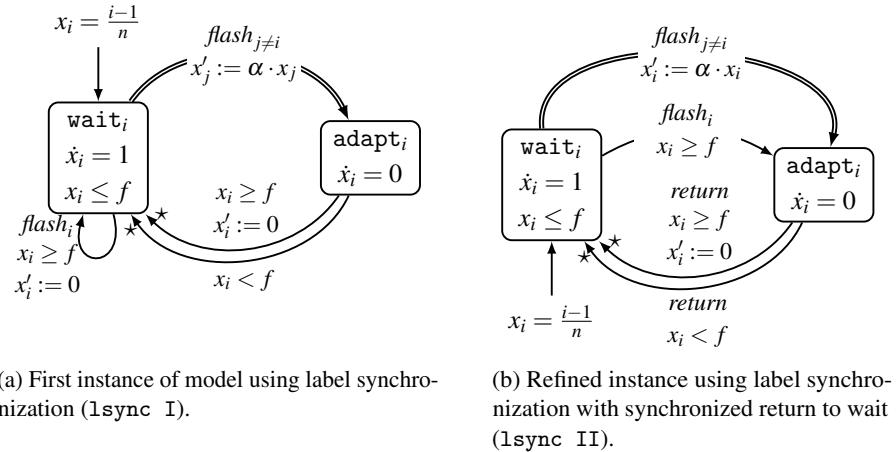


Figure 3: Models of one robot in the synchronization benchmark using label synchronization; jumps marked with a star are urgent, double arrows represent a set of synchronizing jumps, one for each  $j \neq i$ .

the continuous dynamics  $\dot{x}_i = 1$ , which applies as long as  $0 \leq x_i \leq f$  for some *firing threshold*  $f \in \mathbb{R}_{>0}$ . When  $x_i = f$ , robot  $i$  flashes its central LED and  $x_i$  is reset to zero by a discrete event. Robots interact by a simple form of pulse coupling: when robot  $i$  flashes, all other robots are pulled towards firing according to the following relation for some  $\alpha \in \mathbb{R}_{>1}$ :

$$x_i = f \implies x_j := \begin{cases} \alpha \cdot x_j & \text{if } \alpha \cdot x_j < f \\ 0 & \text{otherwise} \end{cases} \quad \text{for all } j \in \{1, \dots, n\} \setminus \{i\} \quad (1)$$

Note that Eq. 1 only describes the update of the clocks – neither the flashing nor implicit flashing of a robots’ LED whenever the clock is reset to 0 is described. These properties have to be added to the model to make them observable. Despite the simple model, the problem of pulse coupling represents a good example of how global swarm behaviors can emerge in distributed systems without being explicitly specified by individual control algorithms. Indeed, a global synchronization of flashing behaviors is achieved – i.e., all clocks are synchronized – even though this is not explicitly imposed by individual controllers.

**Modeling.** We propose several approaches on how to model the synchronization problem compositionally. Each robot is modeled by a hybrid automaton  $\mathcal{H}_i$  such that the swarm behavior for a swarm of size  $n$  is modeled by a hybrid automaton  $\mathcal{H}$  obtained by parallel composition  $\mathcal{H} = \mathcal{H}_1 || \dots || \mathcal{H}_n$  of the single components. In the following we will discuss the different approaches in detail. All approaches model the case distinction of Eq. 1 via separate jumps to reflect the respective clock updates.

**Label synchronization.** The first hybrid automaton model for a single robot  $r_i$  is shown in Figure 3a. It has two control modi `waiti` and `adapti`. To model the system using *label synchronization*, we introduce synchronization labels  $flash_i$ ,  $i = 1, \dots, n$ . Initially, the clocks  $x_i$  of the robots  $r_i$  start with different values in the location `waiti`. When the clock valuation of robot  $r_i$  reaches  $f$ , in the respective automaton a transition with label  $flash_i$  gets enabled; the invariant assures that time cannot further evolve. If the jump with label  $flash_i$  is taken,  $r_i$  resets  $x_i$  to 0 and returns to its `waiti` mode, all other robots  $r_j$ ,  $j \neq i$  synchronize and take their jump with the label  $flash_i$  simultaneously to their `adaptj` mode. Note that for each robot  $r_j$ , there is a jump from `waitj` to `adaptj` with label  $flash_i$  for each  $i \neq j$ , we denote this

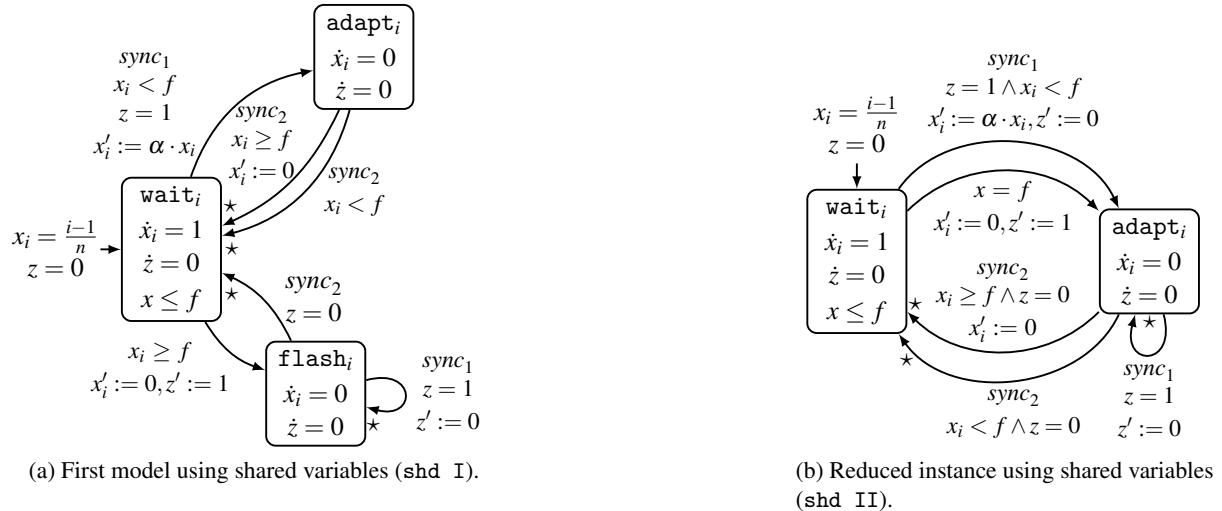


Figure 4: Hybrid automata modelling a single robot in the synchronization benchmark using a shared variable  $z$  for synchronization. Urgent transitions are marked with a star.

in Figure 3a by a double-lined arrow. Due to enabled urgent jumps back to  $wait_j$ , time cannot pass in  $adapt_j$ , but all robots  $r_j$ ,  $j \neq i$  will return using one of their two jumps, modeling the two cases in Eq. 1.

While this model for a single component is relatively simple, using a large number of synchronizing transitions has a strong impact on the size of the parallel composition, as the number of transitions drastically increases with the number of robots (see Table 1). Especially, after synchronizing on  $flash_i$ , all robots  $r_j$ ,  $j \neq i$  will return from  $adapt_j$  to  $wait_j$  before time can further pass. However, since this returning is not synchronized, it can happen in all possible interleaving order, which adds unnecessary complexity. As an improvement, we could apply *partial order reduction* by introducing a fixed order of execution for returning. However, in this special case we can even synchronize all returning transitions, which results in the improved model 1sync II shown in Figure 3b. In this model, we implement the desired behavior by making the returning jumps synchronized on the label *return*. Note that such simplifications need to be carefully designed to assure semantical equivalence, and are hard to automate. In our example, since robot  $r_i$  will also have the label *return*, it also needs to move to  $adapt_i$  in order not to block the others, and will return using the jump with guard  $x_i \geq f$ .

**Shared variables.** Our previous models have two main drawbacks: Firstly, they have a large number of jumps. Secondly, modeling a robot requires full information about the total number of components (and their respective synchronization labels), which does not allow for a generic approach. To achieve a model with less jumps and where single components do not require any prior knowledge about the full system, we make use of *shared variables*. The idea is not to distinguish on which of the robots flashes, but use a shared flag  $z$  to model the fact that *at least one* of the robots flashes. Our first model using shared variables is shown in Figure 4a. Starting with  $z = 0$  initially, once the clock  $x_i$  of robot  $r_i$  reaches the threshold  $f$  it flashes, which is modeled by a jump from  $wait_i$  to a new location  $flash_i$ , which sets  $z$  to 1 to model that a flashing took place. This location has an enabled urgent jump, thus time cannot proceed further. All other robots that flash at the same time need to move to their  $flash$  mode first, in order to be able to synchronize among all the robots on the label  $sync_1$ ; the execution of these synchronized steps adapt the clocks of the non-flashing robots (that move to their  $adapt$  mode) and set  $z$  back to 0 (on the self-loop of the  $flash_i$  mode). A second step synchronizes again all robots, this time on the label  $sync_2$ , to get back to their  $wait$  mode.

Table 1: Number of locations (#locs.) and transitions (#trans.) in the resulting automata for different numbers of robots (#robots).

		#robots							
	version	1	2	3	4	5	6	7	8
#locs.	lsync I	2	3	7	15	31	63	127	255
	lsync II	2	3	4	5	6	7	8	9
	shd I	3	7	15	31	63	127	255	511
	shd II	2	4	8	16	32	64	128	256
#trans.	lsync I	3	6	33	164	755	3310	14077	58728
	lsync II	3	6	15	36	85	198	455	1032
	shd I	6	18	54	162	486	1458	4374	13122
	shd II	5	13	35	97	275	793	2315	6817

Note how the use of a shared variable allowed us to abstract away from the identity of the flashing robot(s): instead of introducing an individual synchronization for the flashing of each of the robots, any flashing robot can set the flag  $z$  and trigger the same synchronization process. Note furthermore that this flag is indeed needed to correctly implement this abstraction: when we would remove it, all robots could adapt (move from waiting to adapting) without any flash happening. Finally, observe that this shared flag allowed to strongly reduce the number of jumps locally in the components as well as globally in the composition, as we do not need to distinguish on the flashing robot's identity any more.

A reduced version shd II, shown in Figure 4b, uses a similar mechanism but unifies the locations  $\text{adapt}_i$  for adaption and  $\text{flash}_i$  for setting the synchronization flag into one location to reduce the parallel composition's size.

To compare our approaches we have created models for the synchronization benchmark for up to 8 robots using all presented approaches. Statistics about the resulting hybrid automata are listed in Table 1. From our results we can observe, that the natural approach via label synchronization (lSync I) creates a high number of transitions in the resulting composed automaton while keeping the number of locations low. Applying the optimization (lSync II) which effectively collects sequences of urgent transitions into a single urgent transition, reduces the number of transitions drastically. The versions using shared variables (shd I + II) produce results with less transitions, e.g., edges for the return to  $\text{wait}$  are collected by a synchronization label ( $\text{sync}_2$ ) and thus create one jump in the parallel composition.

**Reachability analysis results.** Figure 5 shows the flowpipes for a system with three robots and parameters chosen to illustrate how the synchronization occurs and that the analysis is working with sets of states rather than single trajectories. The right image emphasizes a phenomenon that arises from the latter fact.

In Table 2 the running times for our experiments with different numbers of robots using the different modeling approaches can be found. The initial clock valuations for robots  $r_i$  were chosen as  $\frac{i-1}{n}$  as indicated in the models.

## 4.2 Scalability Improvements

Our results so far are unsatisfactory in two ways: first, the number of robots is limited to at most seven and second, the jump depth limit of 20 does not allow to verify that synchronization across all robots occurs. In Sections 4.2.1 and 4.2.2 we apply two optimizations to increase the number of robots the analysis can

Table 2: Running times for different numbers of robots using  $f = 1, \alpha = 1.1$  (time step size:  $\delta = 0.01$ , jump depth: 20, state set representation: boxes). Timeout (TO) is 2 minutes.

version	# robots							
	1	2	3	4	5	6	7	8
lsync I	0.12	<b>0.11</b>	0.13	0.21	1.02	64.6	TO	TO
lsync II	0.12	<b>0.11</b>	<b>0.12</b>	<b>0.14</b>	<b>0.25</b>	<b>2.96</b>	<b>146</b>	TO
shd I	<b>0.11</b>	<b>0.11</b>	<b>0.12</b>	0.16	0.48	9.88	TO	TO
shd II	<b>0.11</b>	<b>0.11</b>	0.13	0.19	0.75	16.85	TO	TO

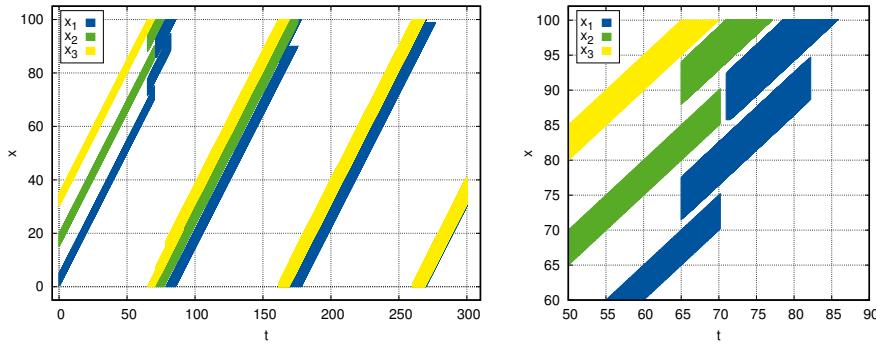


Figure 5: Overlayed flowpipes for a system of 3 robots for  $f = 100, \alpha = 1.1$  (local time horizon 110 sec.). Right: excerpt showing clock adaption after flashing.

handle and then in Section 4.2.3 we optimize precision to allow us to increase jump depth and verify synchronization for a limited number of robots.

We focus on the lsync II model because with the label synchronization approach, synchronization is limited to discrete jumps via labels which we believe increases our chances of success. Focusing on this model allows for several improvements as we show in the next sections. We briefly describe the techniques we use and show intermediate results to illustrate the resulting effect. Each technique is independent from the others and is generalizable for the analysis of any linear hybrid automaton, although the effectiveness depends on the structure of the input. In the following experiments, first we have used unique initial states to examine scalability and to analyze the source of imprecision; in Section 4.2.3 we will extend our experiments to sets of initial states.

#### 4.2.1 Compositional Analysis

Instead of computing the parallel composition of the input automata for each robot eagerly, our aim is to perform an on-the-fly, decomposed analysis. We start with only the initial states and then lazily compute the reachable parts of the automaton as we discover them. This helps to cope with the size of the resulting automaton, as we do not construct locations that are not reached within the bounded analysis and only construct transitions from reachable locations.

We also compute the continuous successors of each automaton individually, i.e., a flowpipe for each automaton where each flowpipe consists of state sets over only the variables of the corresponding automaton, similar to our previous approach in [37]. In the specific case of the synchronization problem, we obtain  $n$  flowpipes as intervals, since each automaton has only one variable. This reduces the dimension

Table 3: Running time in seconds and number of nodes in reachability tree for  $n$  robots with  $f = 1$ ,  $\alpha = 1.3$ , time step size  $\delta = 10^{-5}$  and max jump depth of 20. Transition enumeration is not optimized. Deduplication (see Section 4.2.3) is used. Timeout (TO) is 2 minutes.

$n =$	1	3	5	7	9	11	13	15	17	19	21	23	25
time	5.21	3.56	4.08	6.69	7.13	6.43	15.5	13.0	9.54	8.51	27.3	36.7	TO
nodes	21	21	21	21	21	21	40	30	21	21	28	21	NA

of the state sets, but also reduces precision as we will see next.

**Precision.** Since we separate the variables of the component automata and operate in separate sub-spaces, we lose information on the relation between the variables. This can be illustrated by the following example.

Assume we are using a representation that can exactly describe the trajectories of linear functions, such as oriented boxes or polytopes. Given two robots from our synchronization problem modeled as in `lsync II`, using a single composed state space built from variables  $x_1, x_2$  and starting with initial valuation  $(0, \frac{1}{2}) \subseteq \mathbb{R}^2$ . After taking a time step of  $\delta = \frac{1}{2}$  all reachable valuations are contained in the set  $\{(x, x + \frac{1}{2}) \mid 0 \leq x \leq \frac{1}{2}\}$ . Intersecting this set with  $x_2 \geq 1$  results in the singleton set  $\{(\frac{1}{2}, 1)\}$ . In contrast to this, when working with separate state spaces, we obtain  $[0, \frac{1}{2}] \times [\frac{1}{2}, 1]$  as the set of reachable valuations. Due to the missing relation between the separate state spaces (in this case the relation  $x_2 = x_1 + \frac{1}{2}$ ), the intersection with  $x_2 \geq 1$  gives  $[0, \frac{1}{2}] \times [1, 1]$ .

Note that here the resulting state sets are the same as when using a box representation in the original, composed state space, but the approach does not allow for the sensible use of more precise representations in our case since the problem is one-dimensional in each sub-space. In higher-dimensional sub-spaces, more precise state set representations may improve precision within each subspace but do not affect the loss of information when synchronizing sub-spaces via Cartesian product.

**Results.** In Table 3 the running times and number of nodes in the generated reachability tree for different swarm sizes are shown. The loss of precision that we previously discussed can contribute to branching in the reachability tree as this potentially allows for discrete non-determinism when several transitions are enabled. More concretely, if there are reachable state sets with a clock value below and above  $f$  in the location `adapt`, then successors of both return transitions must be considered, which causes exponential branching behavior. To circumvent this, we use a very small time step of  $\delta = 10^{-5}$ —nonetheless some branching still occurs, which is indicated by node numbers larger than 21.

The compositional approach has already improved the scalability of the analysis, but an exponential number of transitions is still enumerated, causing running times to increase beyond feasibility for larger time horizons or more robots, even when using small initial state sets or even point-sets. Next we look at a way to avoid this exponential explosion.

#### 4.2.2 Optimized Transition Enumeration

Until now, scalability has been impeded by the exponential number of transitions leading from the composed `adapt` location to the `wait` location. Each of the  $n$  automata may take either of its two return transitions, which leads to  $2^n$  transitions in the composed system. However, when considering the actual reachable states of the automata, we observe that in almost all cases only one of the transitions is enabled

Table 4: Running time in seconds and number of nodes in reachability tree for  $n$  robots with  $f = 1$ ,  $\alpha = 1.3$ ,  $\delta = 10^{-5}$  and max jump depth of 20. Transition enumeration is optimized. Deduplication (see Section 4.2.3) is used.

$n =$	25	50	75	100	125	150	175	200	225	250	275	300	325	350	375	400
time	13.6	3.67	4.18	3.97	4.04	6.34	7.13	13.6	4.44	5.16	9.54	19.0	19.5	23.3	12.5	33.1
nodes	21	22	22	22	22	27	30	22	22	22	41	40	40	46	30	26

for each automaton. As we are constructing the jump successors lazily, i.e., we construct them after computing the continuous successors in form of a flowpipe, the information needed to check which transitions are enabled is available to us.

To use this information we chose an arbitrary, but fixed order for the automata. Based on this order, we iteratively determine for each of the transitions (in our example the two return transitions) which flowpipe segments enable the jump in the current automaton. For each transition with one or more enabling segments, we then check for the next automaton which transitions the corresponding segments (synchronized by time) of its flowpipe enable to iteratively restrict the number of segments enabling the considered jump. We enumerate all combinations of transitions with enabling segments in all automata, but do not consider combinations for which there is not a segment enabling the respective jump in each automaton.

In our case this means that we construct exactly one transition from the composed adapt location to the composed wait location, unless there is branching due to loss of precision, as mentioned above.

**Results.** In Table 4 we see the results of the same experiment as before, but with the transition enumeration optimization applied. Clearly, the scalability has greatly improved. Notably, the branching behavior has not fundamentally changed and the maximal jump depth is still set to 20, meaning we can not verify synchronization. Increasing the jump depth leads to more branching because more jumps are made and also because precision keeps decreasing with every jump as over-approximation errors accumulate.

#### 4.2.3 Explicit Time Dimension and Successor Deduplication

To recover the information on the correlation of variable valuations in the different automata, we introduce a variable in each automaton that tracks the time, i.e., its initial value is zero and its derivative is one. When determining the sets of states that enable a transition on which multiple automata synchronize, which is true for all transitions in our example, we know that the time at which the transition is taken must be the same in all automata. We use this by projecting the enabling sets of all synchronizing automata on their time dimension to obtain a time interval during which the transition is enabled. We then intersect these intervals, since all automata must be allowed to jump at the same time, and intersect the enabling sets of the automata with the resulting time interval. For this to be effective, we use a representation that can precisely describe the relation between time and each automaton’s own variable—here we choose template polyhedra with an octagonal template (i.e., polyhedra in half-space representation where the normal vectors are fixed).

For our example from Section 4.2.1 this means that for the automaton starting at  $x_1 = \frac{1}{2}$ , we obtain the point interval  $[\frac{1}{2}, \frac{1}{2}]$  for the time during which the transition with guard  $x \geq 1$  is enabled. Intersecting the other automaton’s enabling segment with this gives us the point interval  $[\frac{1}{2}, \frac{1}{2}]$  for its value of  $x_2$ , i.e., we are able to fully recover the lost information on the relation of the variables.

Table 5: Running time in seconds and number of nodes in reachability tree for  $n$  robots with  $f = 1$ ,  $\alpha = 1.3$ ,  $\delta = 0.01$  and termination once synchronization was detected. We used transition enumeration optimization, transition deduplication and an explicit time dimension.

$n =$	40	80	120	160	200	240	280
time	24.08	47.79	81.56	80.45	104.1	112.4	232.1
nodes	52	54	58	48	56	52	58

**Successor deduplication.** We can now increase the jump depth to verify the synchronization of the robots’ clocks. But the non-determinism that occurs when the clocks do actually synchronize still limits scalability: Since multiple robots may flash at the exact same time and the transition synchronizes on the flash-event of a specific robot  $i$ , each of the automata can take the transition with label  $\text{flash}_i$ , i.e., represents the robot that triggers the flash, or “follows” another robot via the transition labeled  $\text{flash}_{j \neq i}$ .

However, it effectively does not matter which automaton takes which transition, since they all end up in `adapt` and their valuations are all modified in the same way. Cases like this can be identified by identical resets on (synchronized) transitions, i.e., several transitions with the same reset function. Note that while this is generalizable, it makes sense to combine transitions in which the enabling segments overlap—otherwise the approach will introduce additional over-approximation.

In our example, since all those transitions are taken during the same time interval and all of those transitions apply the same reset function, applying the reset function on the union of segments enabling the transitions has the same effect as applying it individually on each enabling segment. As a result, we can resolve the ambiguity between robots which flash simultaneously and produce only one discrete successor which significantly reduces the branching in the resulting reachability tree.

**Results.** In Table 5 we have changed the time step, as it does not need to be as small as before to counteract imprecision. We also removed the cap on jumps and instead terminate when we detect that all automata are synchronized. Thus the number of nodes corresponds to twice the number of flashes needed to establish synchronization, as each flash requires taking two jumps (from `wait` to `adapt` and back).

It is clear that the more sophisticated representation of template polyhedra and overhead of projections increase running times by a noticeable factor, but the desired property of clock synchronization can still be shown for large swarms in reasonable time.

Note that until now we have used point-sets for each clock of each automaton as our first goal was to increase scalability of the general reachability analysis for composed systems and to understand the reasons for the loss of precision and exponential blow-up caused by compositional analysis. As we could show synchronization for these cases after adding several features based on our observations, we next ran our experiments with a setup where the clock valuations of the automata are taken from intervals. The results for different widths of initial sets and different values for  $\alpha$  are shown in Table 6 using breadth-first search and in Table 7 using depth-first search.

We can see that while our previous improvements seemed promising in the sense that they reduced the exponential blow-up in discrete jumps caused by the compositional analysis, the introduced non-determinism from reasonably-sized initial sets still poses a large problem when trying to prove synchronization via flowpipe-construction-based reachability analysis. From the tables we can see, that the non-determinism results in large search trees, which render the analysis infeasible—the running time for single nodes seems negligible, since runs where the search tree has a reasonable size still terminate in less than a second.

Table 6: Running time in seconds and number of nodes in reachability tree for 1 – 3 robots with  $f = 1$ , time step size  $\delta = 10^{-5}$  and max jump depth of 10000 using breadth-first search. Transition enumeration is optimized. Deduplication (see Section 4.2.3) is used.

robots width	1				2				3			
	ver.	dpth.	nodes	time	ver.	dpth.	nodes	time	ver.	dpth.	nodes	time
$\alpha = 1.3$												
1/100	1	1	2	0	1	17	18	0	0	40	12721	60
1/50	1	1	2	0	1	17	26	0	0	38	12899	60
3/100	1	1	2	0	1	21	54	0	0	32	14179	60
$\alpha = 1.2$												
1/100	1	1	2	0	1	29	30	0	0	70	11095	60
1/50	1	1	2	0	1	33	66	0	0	46	12050	60
3/100	1	1	2	0	1	37	230	0	0	34	13468	60
$\alpha = 1.1$												
1/100	1	1	2	0	1	77	192	0	0	68	10625	60
1/50	1	1	2	0	1	101	1055	0	0	55	11022	60
3/100	1	1	2	0	1	120	13918	0	0	49	12046	60

Table 7: Running time in seconds and number of nodes in reachability tree for 1 – 3 robots with  $f = 1$ , time step size  $\delta = 10^{-5}$  and max jump depth of 10000 using breadth-first search. Transition enumeration is optimized. Deduplication (see Section 4.2.3) is used.

robots width	1				2				3			
	ver.	dpth.	nodes	time	ver.	dpth.	nodes	time	ver.	dpth.	nodes	time
$\alpha = 1.3$												
1/100	1	1	2	0	1	17	18	0	0	161	9144	60
1/50	1	1	2	0	1	17	26	0	0	137	9252	60
3/100	1	1	2	0	1	21	54	0	0	221	9357	60
$\alpha = 1.2$												
1/100	1	1	2	0	1	29	30	0	0	289	8803	60
1/50	1	1	2	0	1	33	66	0	0	231	9155	60
3/100	1	1	2	0	1	37	230	0	0	211	8883	60
$\alpha = 1.1$												
1/100	1	1	2	0	1	77	192	0	0	113	7853	60
1/50	1	1	2	0	1	101	1055	0	0	133	7562	60
3/100	1	1	2	0	1	341	11436	0	0	111	7258	60

## 5 Conclusion

In this paper we have shown how algorithms and tools for the verification of hybrid systems could be employed to analyze controllers for swarm robotics. Reachability analysis via flowpipe-construction was used as the basis for the verification of global swarm behavior. Experimental results show the potential of

our approach, nevertheless, several challenges are yet to be addressed in order to increase the applicability of such technique to robotics. Modelling proved to be a challenging task: as observed in our case study, even for a simple system there are manifold ways to create a model which reflects its behavior. While theoretically equivalent, our results show a strong impact of the modeling on the scalability of the analysis of the resulting model. In this context, we have shown several improvements which can be generalized to other problems which allow to partially overcome the scalability issues related to compositional models. Our experimental evaluation indicates the strong effect of those optimizations in a well-controlled environment and even allow to prove properties in some cases. However, our results also indicate, that there is room for improvement, especially when we want to go beyond computing reachability in an over-approximative way but instead want to use results to prove properties for setups which are less controlled (i.e., exhibit more non-determinism, here reflected by larger initial configurations). We hope this work will stimulate further investigations in this exciting and uncharted direction.

**Future work.** Based on our case study we have shown approaches to improve scalability for compositional models. Several ways of further improving our method can be foreseen.

One direction for future work aims at implementing the lazy construction of locations and transitions while still using a single unified state space or potentially even clustering automata into several state spaces. This would allow us to effectively use precise representations while still dealing well with a large number of locations and transitions. We see that the optimizations as shown in Section 4.2.2 can be generalized also for non-decomposed state spaces in compositional models which may represent a middle-ground between running time and precision for composed models using label synchronization. Furthermore, in this case study we solely focused on label synchronization while shared variables were neglected. Investigating dedicated techniques in this area may be another interesting thread of research, for instance, handling shared discrete variables separately may already prove useful.

## References

- [1] Matthias Althoff & John M. Dolan (2014): *Online Verification of Automated Road Vehicles using Reachability Analysis*. *IEEE Transaction on Robotics* 30(4), pp. 903–918, doi:10.1109/tro.2014.2312453.
- [2] Ralph Beckers, Owen E Holland & Jean-Louis Deneubourg (2000): *From Local Actions to Global Tasks: Stigmergy and Collective Robotics*. In: *Prerational Intelligence: Interdisciplinary Perspectives on the Behavior of Natural and Artificial Systems*, 2, Springer, pp. 1008–1022, doi:10.1007/978-94-010-0870-9\_63.
- [3] Michael Bonani, Valentin Longchamp, Stéphane Magnenat, Philippe Réturnaz, Daniel Burnier, Gilles Roulet, Florian Vaussard, Hannes Bleuler & Francesco Mondada (2010): *The MarXbot, a Miniature Mobile Robot Opening New Perspectives for the Collective-robotic Research*. In: *Proc. of IROS’10*, IEEE, pp. 4187–4193, doi:10.1109/iros.2010.5649153.
- [4] Manuele Brambilla, Eliseo Ferrante, Mauro Birattari & Marco Dorigo (2013): *Swarm Robotics: a Review from the Swarm Engineering Perspective*. *Swarm Intelligence* 7(1), pp. 1–41, doi:10.1007/s11721-012-0075-2.
- [5] Manuele Brambilla, Carlo Pinciroli, Mauro Birattari & Marco Dorigo (2012): *Property-driven Design for Swarm Robotics*. In: *Proc. of AAMAS’12*, IFAAMAS, pp. 139–146.
- [6] John Buck (1988): *Synchronous Rhythmic Flashing of Fireflies*. *The Quarterly Review of Biology* 63(3), pp. 265–289, doi:10.1086/394562.
- [7] Xin Chen, Erika Abraham & Sriram Sankaranarayanan (2012): *Taylor Model Flowpipe Construction for Non-linear Hybrid Systems*. In: *Proc. of RTSS’12*, IEEE, pp. 183–192, doi:10.1109/rtss.2012.70.
- [8] Xin Chen, Erika Abraham & Sriram Sankaranarayanan (2013): *Flow\*: An Analyzer for Non-linear Hybrid Systems*. In: *Proc. of CAV’13, LNCS 8044*, Springer, pp. 258–263, doi:10.1007/978-3-642-39799-8\_18.

- [9] Frederick Ducatelle, Gianni A. Di Caro, Carlo Pinciroli & Luca Maria Gambardella (2011): *Self-organized Cooperation between Robotic Swarms*. *Swarm Intelligence* 5(2), pp. 73–96, doi:10.1007/s11721-011-0053-0.
- [10] Eliseo Ferrante, Manuele Brambilla, Mauro Birattari & Marco Dorigo (2010): *Socially-Mediated Negotiation for Obstacle Avoidance in Collective Transport*. In: *Proc. of DARS’10, Springer Tracts in Advanced Robotics* 83, Springer, pp. 571–583, doi:10.1007/978-3-642-32723-0\_41.
- [11] Goran Frehse (2015): *An Introduction to Hybrid Automata, Numerical Simulation and Reachability Analysis*. In: *Formal Modeling and Verification of Cyber-Physical Systems*, Springer, pp. 50–81, doi:10.1007/978-3-658-09994-7\_3.
- [12] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang & Oded Maler (2011): *SpaceEx: Scalable verification of hybrid systems*. In: *Proc. of CAV’11, LNCS* 6806, Springer, pp. 379–395, doi:10.1007/978-3-642-22110-1\_30.
- [13] Antoine Girard (2005): *Reachability of Uncertain Linear Systems Using Zonotopes*. In: *Proc. of HSCC’05, LNCS* 3414, Springer, pp. 291–305, doi:10.1007/978-3-540-31954-2\_19.
- [14] Thomas A. Henzinger (1996): *The Theory of Hybrid Automata*. In: *Proc. of LICS’96*, IEEE, pp. 278–292, doi:10.1007/978-3-642-59615-5\_13.
- [15] *HyCreate*. Available at <http://stanleybak.com/projects/hycreate/hycreate.html>.
- [16] Savas Konur, Clare Dixon & Michael Fisher (2012): *Analysing Robot Swarm Behaviour via Probabilistic Model Checking*. *Robotics and Autonomous Systems* 60(2), pp. 199–213, doi:10.1016/j.robot.2011.10.005.
- [17] Panagiotis Kouvaros & Alessio Lomuscio (2015): *A Counter Abstraction Technique for the Verification of Robot Swarms*. In: *Proc. of AAAI’15*, AAAI Press, pp. 2081–2088.
- [18] Panagiotis Kouvaros & Alessio Lomuscio (2015): *Verifying Emergent Properties of Swarms*. In: *Proc. of IJCAI’15*, AAAI Press, pp. 1083–1089.
- [19] Alex A. Kurzhanskiy & Pravin Varaiya (2007): *Ellipsoidal Techniques for Reachability Analysis of Discrete-Time Linear Systems*. *IEEE Transactions on Automatic Control* 52(1), pp. 26–38, doi:10.1109/tac.2006.887900.
- [20] Thomas Halva Labella, Marco Dorigo & Jean-Louis Deneubourg (2004): *Efficiency and Task Allocation in Prey Retrieval*. In: *Proc. of BioADIT’04, LNCS* 3141, Springer, pp. 274–289, doi:10.1007/978-3-540-27835-1\_21.
- [21] Colas Le Guernic & Antoine Girard (2010): *Reachability Analysis of Linear Systems using Support Functions*. *Nonlinear Analysis: Hybrid Systems* 4(2), pp. 250–262, doi:10.1016/j.nahs.2009.03.002.
- [22] Wenguo Liu, Alan F. T. Winfield, Jin Sa, Jie Chen & LiHua Dou (2006): *Strategies for Energy Optimisation in a Swarm of Foraging Robots*. In: *Proc. of SAB’06, LNCS* 4433, Springer, pp. 14–26, doi:10.1007/978-3-540-71541-2\_2.
- [23] Alessio Lomuscio & Edoardo Pirovano (2018): *Verifying Emergence of Bounded Time Properties in Probabilistic Swarm Systems*. In: *Proc. of IJCAI’18, ijcai.org*, pp. 403–409, doi:10.24963/ijcai.2018/56.
- [24] Alessio Lomuscio & Edoardo Pirovano (2019): *A Counter Abstraction Technique for the Verification of Probabilistic Swarm Systems*. In: *Proc. of AAMAS’19*, International Foundation for Autonomous Agents and Multiagent Systems, pp. 161–169. Available at <http://hdl.handle.net/10044/1/20101>.
- [25] Maja J. Mataric (1993): *Designing Emergent Behaviors: From Local Interactions to Collective Intelligence*. In: *Proc. of SAB’93*, pp. 432–441.
- [26] Renato E Mirolo & Steven H Strogatz (1990): *Synchronization of Pulse-coupled Biological Oscillators*. *SIAM Journal on Applied Mathematics* 50(6), pp. 1645–1662, doi:10.1137/0150098.
- [27] Francesco Mondada, Giovanni C. Pettinari, André Guignard, Ivo W. Kwee, Dario Floreano, Jean-Louis Deneubourg, Stefano Nolfi, Luca Maria Gambardella & Marco Dorigo (2004): *Swarm-Bot: A New Distributed Robotic Concept*. *Autonomous Robots* 17(2-3), pp. 193–221, doi:10.1023/b:auro.0000033972.50769.1c.
- [28] Ramon E. Moore, Ralph Baker Kearfott & Michael J. Cloud (2009): *Introduction to Interval Analysis*. SIAM, doi:10.1137/1.9780898717716.
- [29] Lynne E. Parker (2000): *Current State of the Art in Distributed Autonomous Mobile Robotics*. In: *Proc. of DARS’00*, Springer, pp. 3–14, doi:10.1007/978-4-431-67919-6\_1.

- [30] Joaquín Peña, Christopher A. Rouff, Mike Hinckey & Antonio Ruiz Cortés (2011): *Modeling NASA Swarm-based Systems: Using Agent-oriented Software Engineering and Formal Methods*. *Software and System Modeling* 10(1), pp. 55–62, doi:10.1007/s10270-009-0135-2.
- [31] Carlo Pincioli, Vito Trianni, Rehan O’Grady, Giovanni Pini, Arne Brutschy, Manuele Brambilla, Nithin Mathews, Eliseo Ferrante, Gianni Di Caro, Frederick Ducatelle, Mauro Birattari, Luca Maria Gambardella & Marco Dorigo (2012): *ARGoS: A Modular, Parallel, Multi-engine Simulator for Multi-robot Systems*. *Swarm Intelligence* 6(4), pp. 271–295, doi:10.1007/s11721-012-0072-5.
- [32] Christopher Rouff, Amy Vanderbilt, Michael G. Hinckey, Walt Truszkowski & James L. Rash (2004): *Properties of a Formal Method for Prediction of Emergent Behaviors in Swarm-Based Systems*. In: *Proc. of SEFM’04*, IEEE, pp. 24–33.
- [33] Erol Sahin (2004): *Swarm Robotics: From Sources of Inspiration to Domains of Application*. In: *Proc. of SAB’04, LNCS 3342*, Springer, pp. 10–20, doi:10.1007/978-3-540-30552-1\_2.
- [34] Sriram Sankaranarayanan, Thao Dang & Franjo Ivančić (2008): *Symbolic Model Checking of Hybrid Systems Using Template Polyhedra*. In: *Proc. of TACAS’08, LNCS 4963*, Springer, pp. 188–202.
- [35] Stefan Schupp (2019): *State Set Representations and their Usage in the Reachability Analysis of Hybrid Systems*. Dissertation, RWTH Aachen University, doi:10.18154/RWTH-2019-08875. Available at <http://publications.rwth-aachen.de/record/767529>.
- [36] Stefan Schupp, Erika Abraham, Ibtissem Ben Makhoul & Stefan Kowalewski (2017): *HyPro: A C++ Library for State Set Representations for Hybrid Systems Reachability Analysis*. In: *Proc. of NFM’17, LNCS 10227*, Springer, pp. 288–294, doi:10.1007/978-3-319-57288-8\_20.
- [37] Stefan Schupp, Johanna Nellen & Erika Abraham (2017): *Divide and Conquer: Variable Set Separation in Hybrid Systems Reachability Analysis*. In: *Proc. of QAPL’17, EPTCS 250*, pp. 1–14, doi:10.4204/eptcs.250.1.
- [38] Thomas Walker (1969): *Synchronous Rhythmic Flashing of Fireflies. Acoustic Synchrony: Two Mechanisms in the Snowy Tree Cricket* 166(3907), pp. 891–894.
- [39] Arthur Winfree (1967): *Biological Rhythms and the Behavior of Populations of Coupled Oscillators*. *Journal of Theoretical Biology* 16(1), pp. 15–42, doi:10.1016/0022-5193(67)90051-3.
- [40] Günter M. Ziegler (1995): *Lectures on polytopes*. 152, Springer Science & Business Media, doi:10.1007/978-3-0348-8438-9\_1.